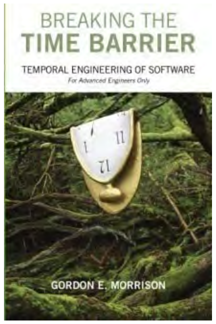
Understanding Temporal Logic

Introducing

Coherent Object System Architecture (COSA)

By
Gordon Morrison, Author
Breaking the Time Barrier



www.vsmerlot.com

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **APR 2010**	2. REPORT TYPE	3. DATES COVERED **00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE **Understanding Temporal Logic Introducing Coherent Object System Architecture (COSA)**	5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **VS Merlot,League City,TX,77573**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Presented at the 22nd Systems and Software Technology Conference (SSTC), 26-29 April 2010, Salt Lake City, UT. Sponsored in part by the USAF. U.S. Government or Federal Rights License

14. ABSTRACT

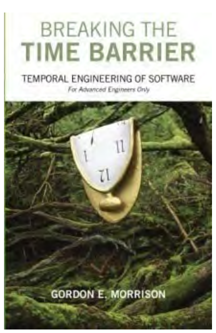
15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**32**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

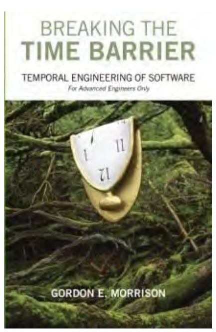
The Challenge

- Using the traditional spatial If-Then-Else (ITE) approach
  - Produce a five-function calculator
  - add, subtract, multiply, divide, and percent
- The specification is at: www.vsmerlot.com
- Count the number of ITE and Case Statements
  - Count every logic decision point
  - Don’t use my temporal COSA approach
  - Did you improve on COSA?



Proper State Machine(1)

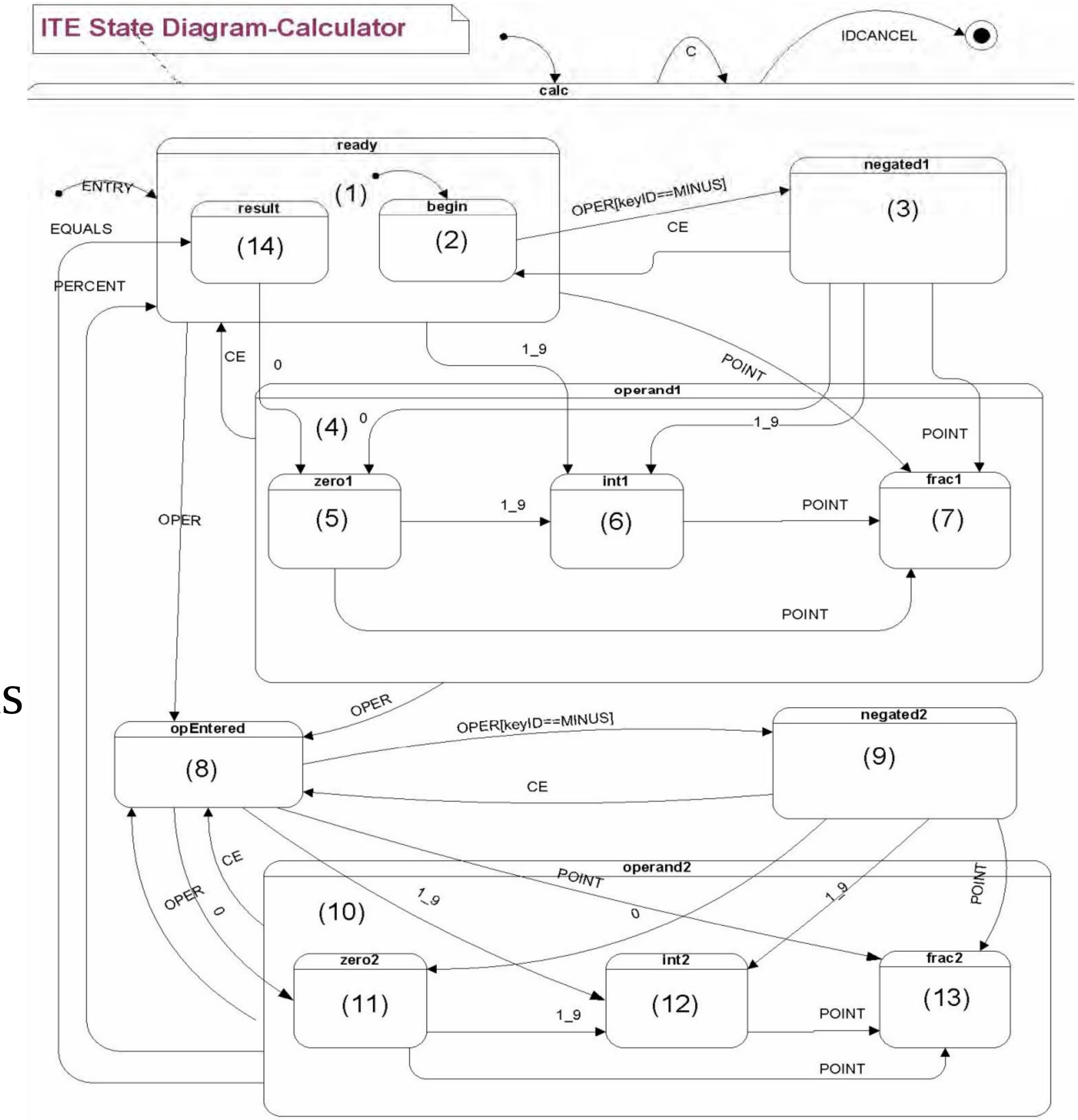
- In a proper state machine, the state transitions are all complete and orthogonal.
  - Complete Transitions: a transition is defined for every possible situation.
  - Orthogonal Transitions: none of the transitions have overlapping conditions.
- With a proper state machine
  - a next state is defined for every possible condition
  - the designated next state is unique
- My comment:
  - The proper state machine is spatial using ITE
  - The proper state machine doesn’t know where it’s working

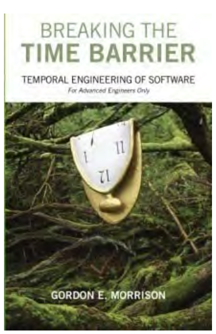


(1)- © 2005 Carnegie Mellon University – PSP II Designing and Verifying State Machines- page 41

ITE Calculator Statechart

- The author's implementation:
  - 112 ITE / Case
    - Ready – 6 case
    - Eval – 4 case + ?
  - 1,000+ LOC
- Arrows represent transitions
  - Transitions are events
- Minus is an ambiguous transition
    - Begin to negate1
    - subtract
    - opEntered to negated2

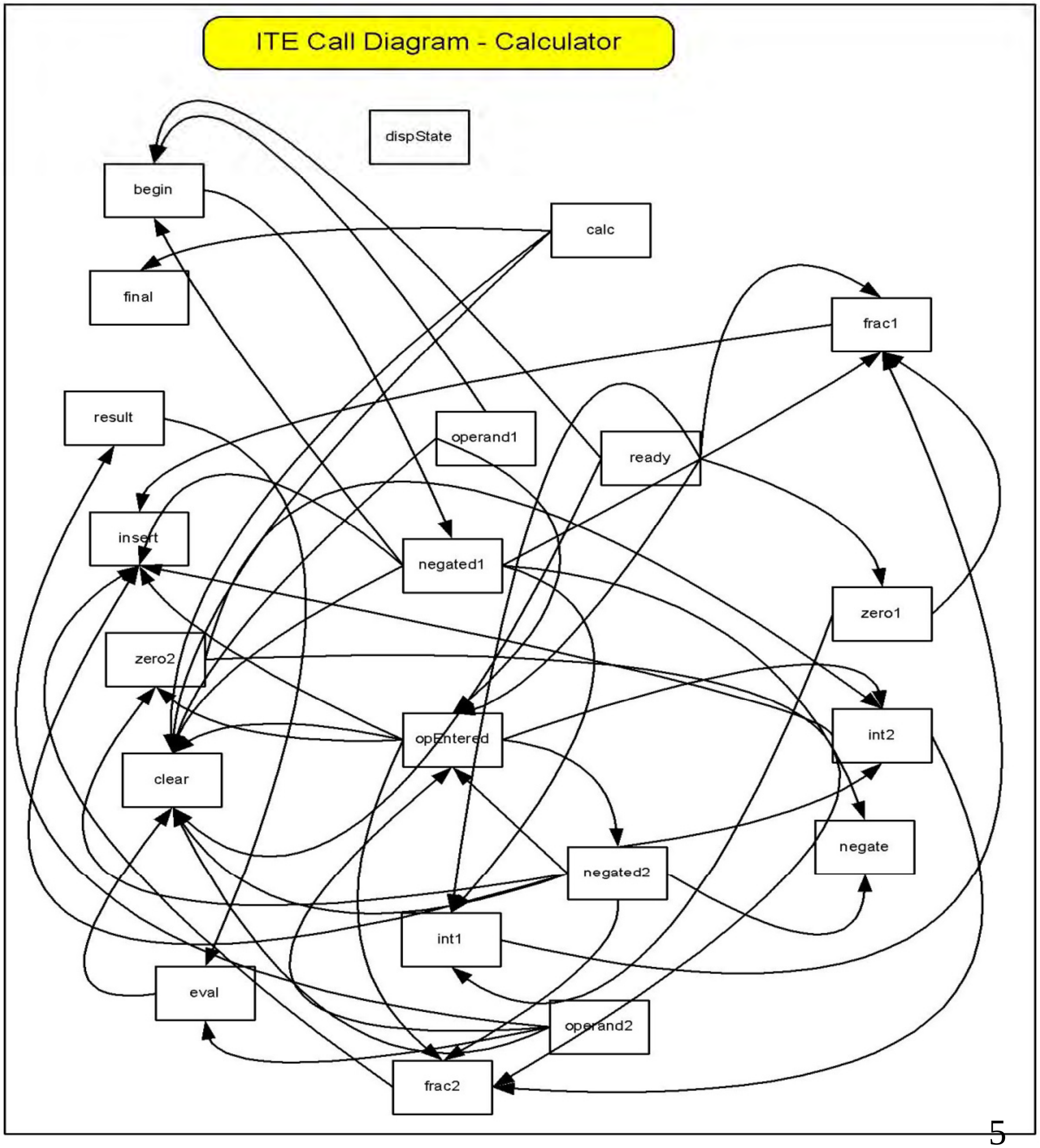


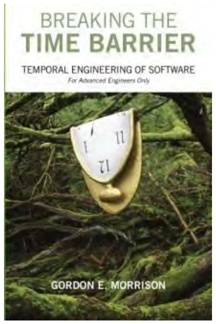


"Practical Statecharts in C/C++, © CMP Books, Miro Samek, Ph.D.

ITE Calculator Call Diagram

- No calls to trace display
- Very complex



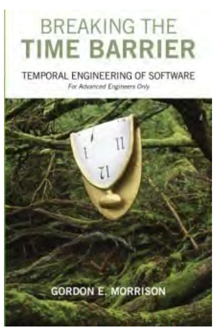


ITE State Machine

Trace debug is everywhere in code.

```
QHsm::CQSTATE Calc1::opEntered(QEvent const *e) {
  switch (e->sig) {
  case Q_ENTRY_SIG:
    dispState("opEntered");
    return 0;
  case IDC_OPER:
     dispState("IDC Entered");
    if (((CalcEvt *)e)->keyId == IDC_MINUS) {
      clear();
      Q_TRAN(&Calc1::negated1);
    }
    return 0;
  case IDC_0:
    dispState("IDC 0 Entered");
    clear();
    Q_TRAN(&Calc1::zero1);
    return 0;
```

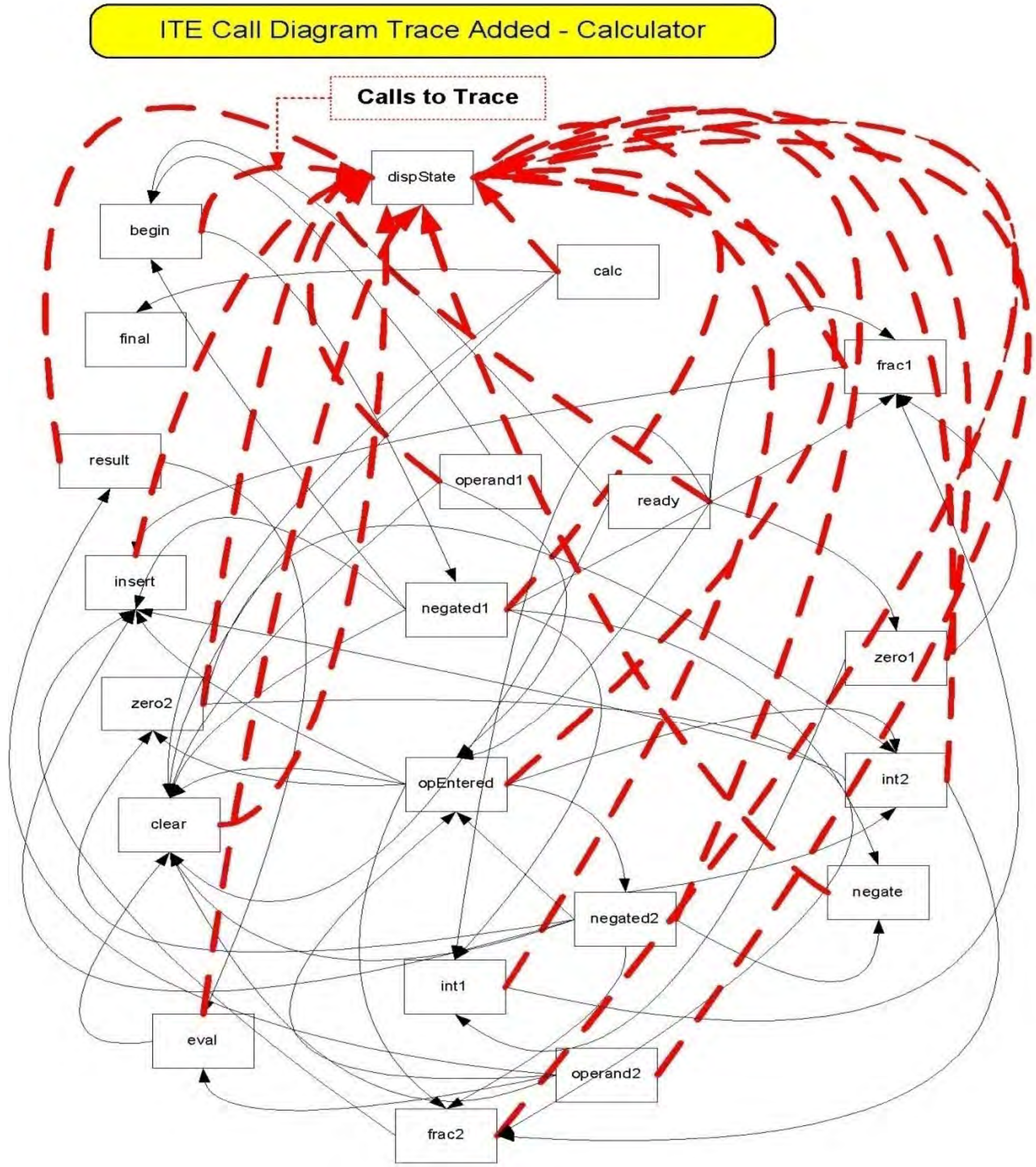
```
case IDC_1_9:
     dispState("IDC 1-9 Entered");
      clear();
      insert(((CalcEvt *)e)->keyId);
      Q_TRAN(&Calc1::int1);
      return 0;
  case IDC_POINT:
      clear();
      dispState("IDC Point Entered");
      insert(IDC_0);
      insert(((CalcEvt *)e)->keyId);
      Q_TRAN(&Calc1::frac1);
      return 0;
  }
  return QSTATE_SC(&Calc1::calc);
}
```

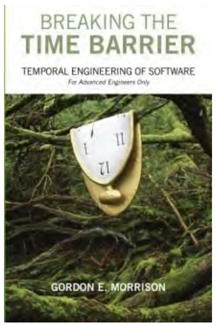


"Practical Statecharts in C/C++, © CMP Books, Miro Samek, Ph.D.

ITE With Debugging

- ITE trace
  - Red lines …
- Trace debug
  - Each function
  - Embedded
  - Side effects

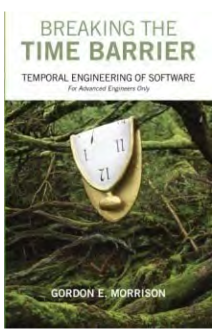




COSA vs. Traditional ITE

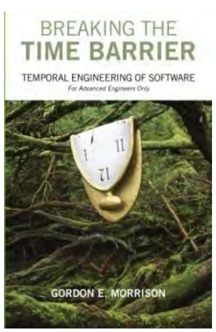
- Temporal domain
  - Time Indexed
- Reduces complexity
  - State diagrams
  - Call diagrams
  - Models
- Reduces code size
- Increases reuse
- Includes trace
- Preemptable

- Spatial domain
  - Find where last
- Increases complexity
  - State diagrams
  - Call diagrams
  - Models
- Increases code size
- Decreases reuse
- Manual trace
- ~Not Preemptable



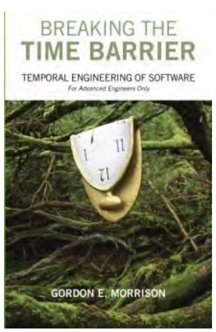
Temporal vs. Spatial

- Imagine a CPU without a program counter (PC)
  - The hardware would need to save states continuously
  - After an interrupt determine where it was executing
  - Massive amount of logic as administrative overhead
  - This is spatial
- The PC is a temporal pointer
- Software does not have an equivalent PC
  - Until COSA was invented (see US Patent)



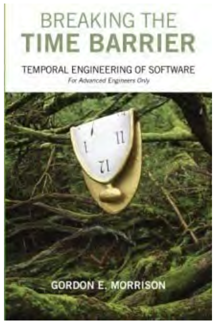
Proper COSA State Machine

- Engine/Table relationship
  - Table contains 1 or more rules
  - Each rule has a single entry point
- Rules consist of steps
- Every step is a binary state
  - Each step has a test condition
    - a True Behavior / Next Rule/Step Transition
    - a False Behavior / Next Rule/Step Transition
    - and a Trace (tied to the specification)



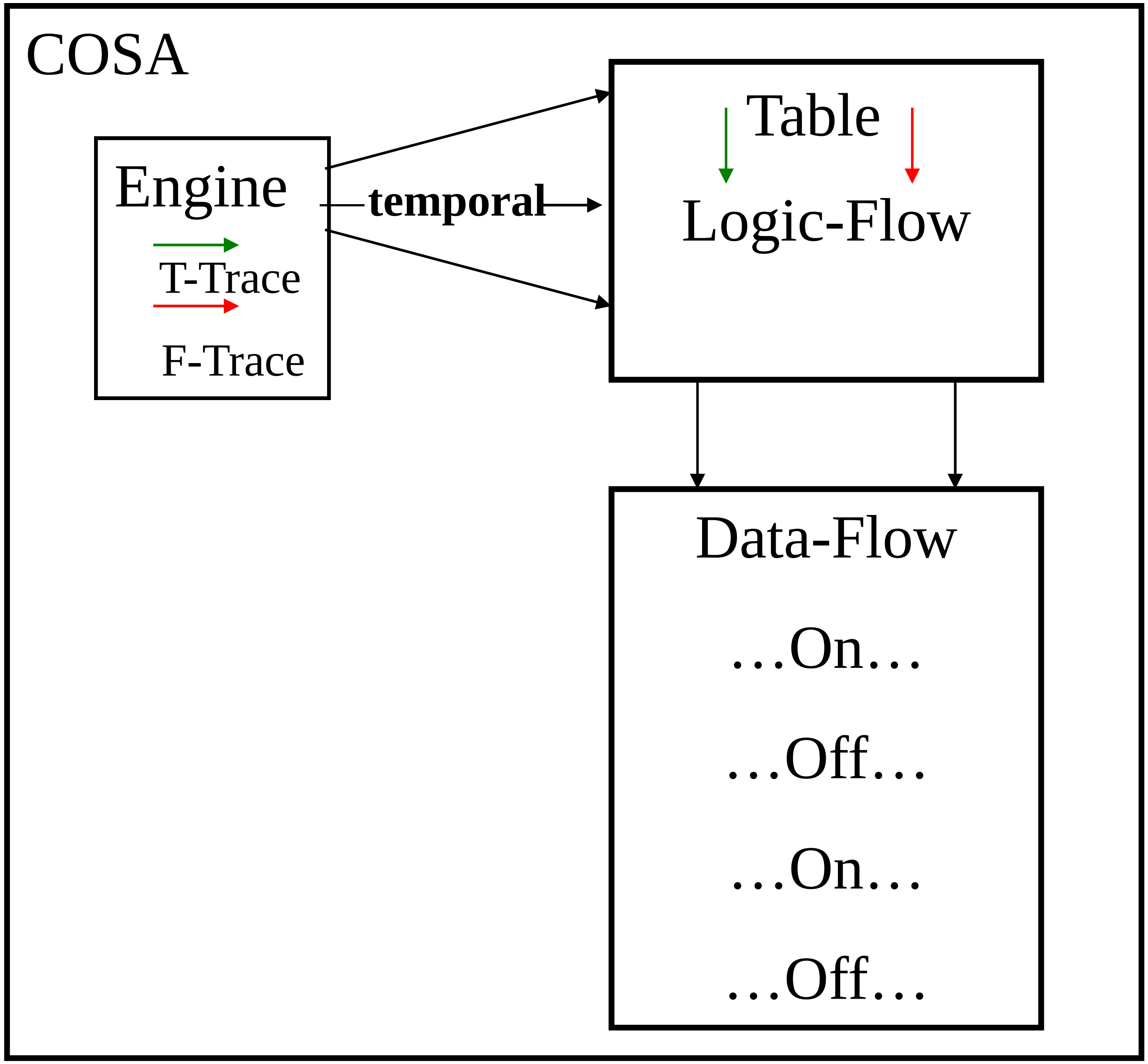
COSA State Machine

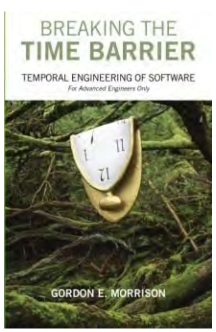
- Event or non-event driven applications
- States are true or false
  - Transitions are next true or next false
- Three fundamental parts
  - Engine (temporal, trace, and control)
  - Logic-Flow / Rule Table (class)
  - Data-Flow / Reusable Members
- Logic-flow is orthogonal to data-flow



COSA Pattern

- One or more engine/table pairs
- Tracing
  - Duo Point
    - True Trace
    - False Trace





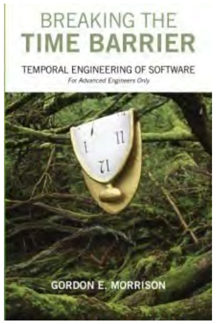
Engine / Table Pattern

Engine

- Engine is temporal (iTime)
- Preemption control
  - Test condition (state)
    - Dynamic bind **True** Behavior
      - Next True Rule/Step
      - True Trace
    - Dynamic bind **False** Behavior
      - Next False Rule/Step
      - False Trace
  - End preemption control loop

Table

- Each row is a temporal sequence
  - Rule/Step (name)
  - Test condition (state)
    - **True** Behavior
      - Next True Rule/Step
    - **False** Behavior
      - Next False Rule/Step
  - Trace (unique to app)

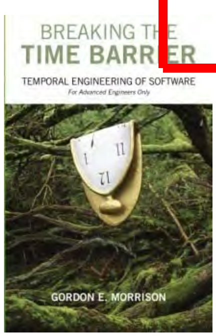


COSA Engine Detail

```
procedure TCOSAFrame.Run(intState integer);
begin
  bEngine := TRUE;
  iState := intState;
  while bEngineLocal AND bEngineGlobal do
  begin
        if iState = rRule[iTime].iState then
        begin
             rRule[iTime].pTrueRule;
             True_Trace(iTime);
             iTime := rRule[iTime].iTrueRule;
        end else
        begin
             rRule[iTime].pFalseRule;
             False_Trace(iTime);
             iTime := rRule[iTime].iFalseRule;
        end;
  end;
end;
```

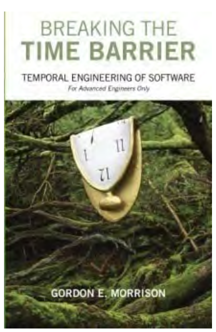
iTime is temporal

Determined by logic



23-Steps of Calculator Logic

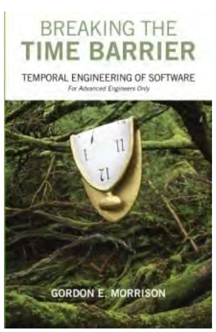
```
//                   Static     True           Next True  False          Next False
//       Rules       State      Behavior          Rule    Behavior          Rule       Trace
  pBRT(rOpr1,    iNeg44,  Negate,       {0} rOpr1+1, Clr_Buf,    {1} rOpr1+1,  100);
  pBRT(rOpr1+1,  iDigit,  Any_Number,   {0} rOpr1+1, Ignore,     {1} rOpr1+2,  101);
  pBRT(rOpr1+2,  iDot59,  One_Period,   {0} rOpr1+3, Ignore,     {1} rOpr1+4,  102);
  pBRT(rOpr1+3,  iDigit,  Any_Number,   {0} rOpr1+3, Ignore,     {1} rOpr1+4,  103);
// clear
  pBRT(rOpr1+4,  iClEnt,  Clear_Entry,{0} rOpr1,   Ignore,     {1} rOpr1+5,  104);
  pBRT(rOpr1+5,  iClear,  Clear,        {0} rOpr1,   Ignore,     {1} rOpr1+6,  105);
  pBRT(rOpr1+6,  iPush,   Push_Disp,    {1} rOpr8,   Push_Disp,  {1} rOpr8,    106);
// operations
  pBRT(rOpr8,    iAdd43,  Addition,     {1} rOpr2,   Ignore,     {1} rOpr8+1,  500);
  pBRT(rOpr8+1,  iSub44,  Subtraction,{1} rOpr2,   Ignore,     {1} rOpr8+2,  501);
  pBRT(rOpr8+2,  iMul42,  Multiply,     {1} rOpr2,   Ignore,     {1} rOpr8+3,  502);
  pBRT(rOpr8+3,  iDiv47,  Division,     {1} rOpr2,   Ignore,     {1} rOpr2,    503);
// next number
  pBRT(rOpr2,    iOff,    Engine_Off,   {0} rOpr2+1, Ignore,     {0} rErr,     700);
  pBRT(rOpr2+1,  iNeg44,  Negate,       {0} rOpr2+2, Ignore,     {1} rOpr2+2,  701);
  pBRT(rOpr2+2,  iDigit,  Any_Number,   {0} rOpr2+2, Ignore,     {1} rOpr2+3,  702);
  pBRT(rOpr2+3,  iDot59,  One_Period,   {0} rOpr2+4, Ignore,     {1} rOpr2+5,  703);
  pBRT(rOpr2+4,  iDigit,  Any_Number,   {0} rOpr2+4, Ignore,     {1} rOpr2+5,  704);
// clear
  pBRT(rOpr2+5,  iClEnt,  Clear_Entry,{0} rOpr2+1, Ignore,     {1} rOpr2+6,  705);
  pBRT(rOpr2+6,  iClear,  Clear,        {0} rOpr1,   Ignore,     {1} rOpr2+7,  706);
  pBRT(rOpr2+7,  iSave,   Save_Disp,    {0} rResu,   Save_Disp,  {1} rResu,    707);
// equals
  pBRT(rResu,    iPer37,  Percent,      {0} rOpr1,   Ignore,     {1} rResu+1,  900);
  pBRT(rResu+1,  iEqual,  Equals,       {0} rOpr1,   SetChain,   {1} rResu+2,  901);
  pBRT(rResu+2,  iChain,  Operate,      {0} rOpr1+6, Error,      {0} rErr,     902);
  pBRT(rErr,     iErr86,  Error,        {0} rOpr1,   Unknown,    {0} rOpr1,    993);
end;
```



The User Perspective

(tends to be temporal)

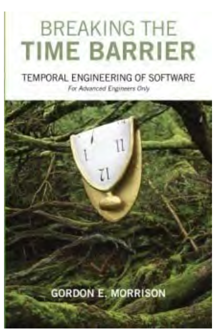
- Calculator
  - Enter Operand 1 (optional sign)
  - Enter Operation (+ - * /)
  - Enter Operand 2 (optional sign)
  - Select Result Type (= % (+ - * /))
- The user perspective is generally temporal
- Enter ‘-’ ‘3’ ‘.’ ‘1’ ‘4’ ‘1’ ‘5’ ‘9’



Understanding the Time Index

ENTER	Rule	State	True Action	Next True	False Action	Next False
‘-’	rOper1	= <iNeg44>?	Negate	rOper+1	Ignore	rOper+1
‘3’	+1	= <iDigit>*	Any_Number	rOper+1	Ignore	rOper+2
‘.’	+2	= <iDot59>?	One_Period	rOper+3	Ignore	rOper1+4
‘14159’	+3	= <iDigit>*	Any_Number	rOper1+3	Ignore	rOper1+4
Time→	+4					

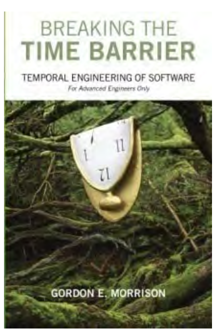
- I know where I am
- I know where I came from
- I know where I am going
- At iTime+4 - Not a number from iTime+3



Logic and Trace

Rule	State	True Action	Next	False Action	Next	Trace
rOper1	iNeg44	Negate	rOper1+1	Ignore	rOper1+1	100
+1	iDigit	Any_Number	rOper1+1	Ignore	rOper1+2	101
+2	iDot59	One_Period	rOper1+3	Ignore	rOper1+4	102
+3	iDigit	Any_Number	rOper1+3	Ignore	rOper1+4	103
Time +4						

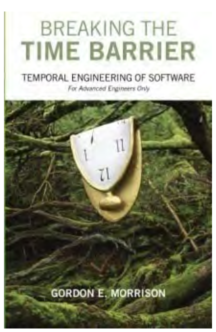
T	TR	DS	Behavior	Value
1	100	44;	Negate;	N= -
2	101	1;	Any_Number;	N= -3
3	101	59;	Ignore;	N=
4	102	59;	One_Period;	N= -3.
5	103	1;	Any_Number;	N= -3.1
6	103	1;	Any_Number;	N= -3.14
7	103	1;	Any_Number;	N= -3.141
8	103	1;	Any_Number;	N= -3.1415
9	103	1;	Any_Number;	N= -3.14159
10	103	44;	Ignore;	N=



Some ITE Logic

```
QHsm::CQSTATE Calc1::opEntered(QEvent const *e) {
  switch (e->sig) {
  case Q_ENTRY_SIG:
    dispState("opEntered");
    return 0;
  case IDC_OPER:
     dispState("IDC Entered");
    if (((CalcEvt *)e)->keyId == IDC_MINUS) {
      clear();
      Q_TRAN(&Calc1::negated1);
    }
    return 0;
  case IDC_0:
    dispState("IDC 0 Entered");
    clear();
    Q_TRAN(&Calc1::zero1);
    return 0;
```

```
case IDC_1_9:
    dispState("IDC 1-9 Entered");
     clear();
     insert(((CalcEvt *)e)->keyId);
     Q_TRAN(&Calc1::int1);
     return 0;
  case IDC_POINT:
     clear();
     dispState("IDC Point Entered");
     insert(IDC_0);
     insert(((CalcEvt *)e)->keyId);
     Q_TRAN(&Calc1::frac1);
     return 0;
  }
  return QSTATE_SC(&Calc1::calc);
}
```

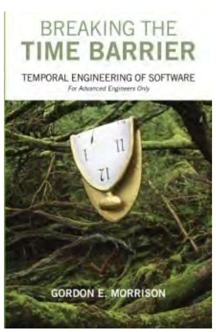


"Practical Statecharts in C/C++, © CMP Books, Miro Samek, Ph.D.

Compare COSA Trace

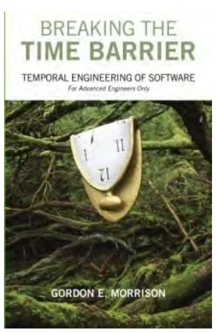
COSA Trace

T	TR	DS	Behavior	Value
1	100	44;	Negate;	N= -
2	101	1;	Any_Number;	N= -3
3	101	59;	Ignore;	N=
4	102	59;	One_Period;	N= -3.
5	103	1;	Any_Number;	N= -3.1
6	103	1;	Any_Number;	N= -3.14
7	103	1;	Any_Number;	N= -3.141
8	103	1;	Any_Number;	N= -3.1415
9	103	1;	Any_Number;	N= -3.14159
10	103	44;	Ignore;	N=



To ITE Trace

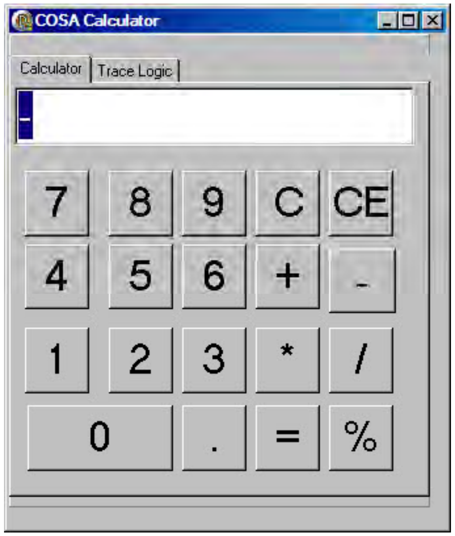
T	Behavior	e->sig	Value
O O O			
19,	g-negated1,	2,	0
20,	**negated1**		
21,	g-negated1,	1,	-0
22,	g-negated1,	1010,	-0
O O O			
31,	**int1**		
32,	g-int1,	1,	-3
33,	g-int1,	1101,	-3
34,	g-frac1,	0,	-3.
O O O			
37,	g-frac1,	2,	-3.
38,	**frac1**		
39,	g-frac1,	1,	-3.
40,	g-frac1,	1010,	-3.
O O O			
45,	g-frac1,	1107,	-3.14159
46,	g-Oper1,	1107,	-3.14159
47,	g-frac1,	3,	-3.14159
48,	g-opEntered,	0,	-3.14159
49,	g-Oper1,	0,	-3.14159
50,	g-Oper1,	3,	-3.14159
51,	g-opEntered,	2,	-3.14159
52,	**opEntered**		
53,	g-opEntered,	1,	-3.14159
54,	g-opEntered,	1107,	-3.14159

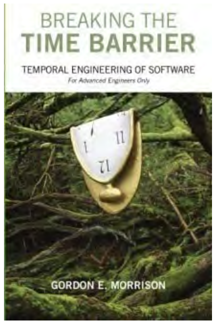


COSA

- -3.14159 – ten steps to enter eight actions
  - 80% efficient
  - 20% of cost is overhead

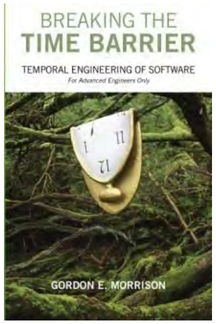
- -3.14159 - - 2.14195 = thirty steps for eighteen actions
  - 60% efficient
  - 40% of cost is overhead





ITE

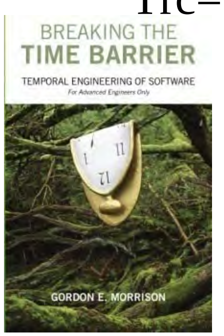
- -3.14159 – fifty-four steps four eight actions
  - 14.8 % efficient
  - 85% of cost is overhead
- -3.14159 - - 2.14195 = 107 steps for eighteen actions
  - 16.8 % efficient
  - 83% of cost is overhead



ITE Enter “-” Only

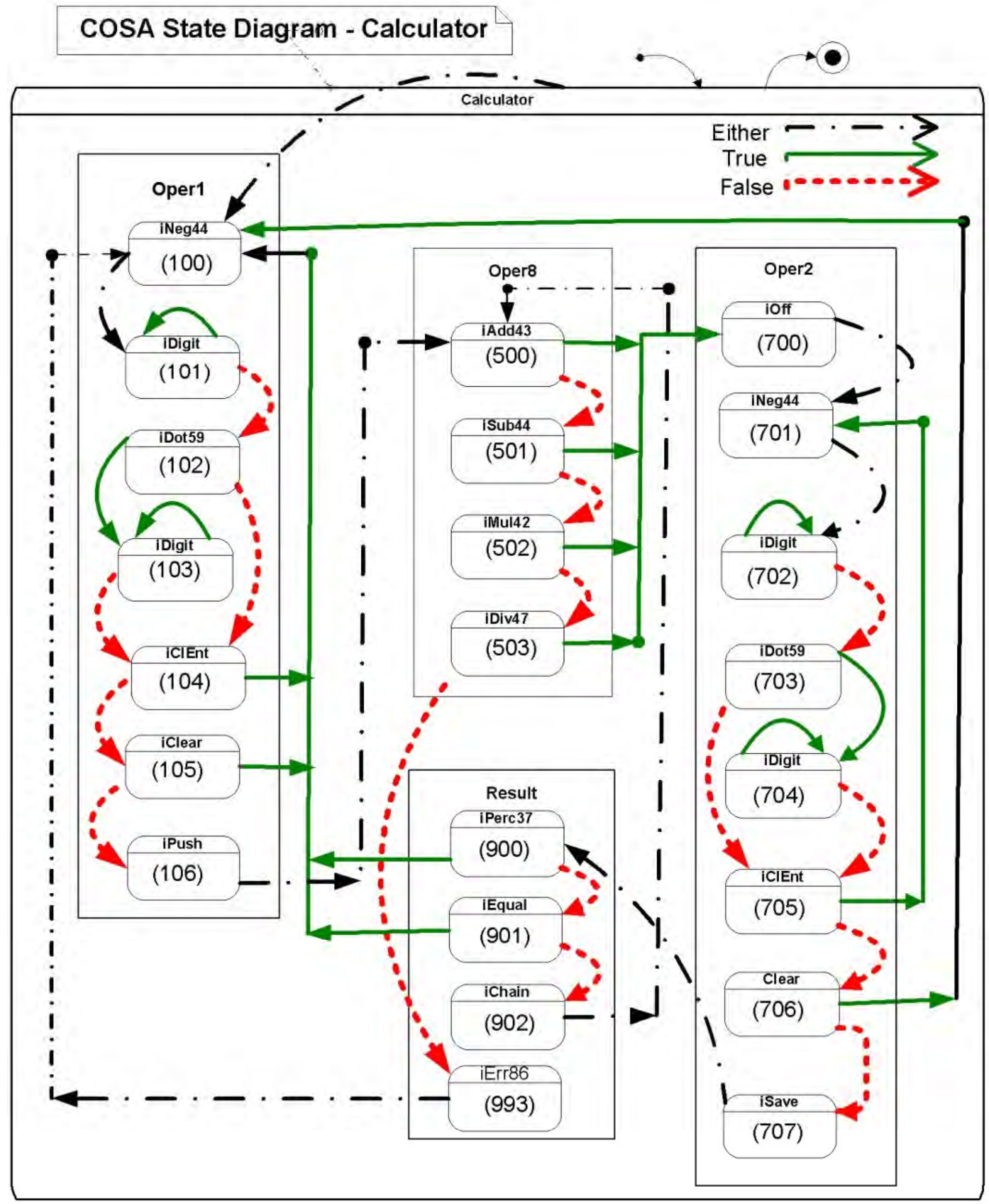
Trc= 1, g-calc, sig= 0 ; Operand= ,
Trc= 2, g-calc, sig= 0 ; Operand= ,
Trc= 3, g-calc, sig= 1 ; Operand= ,
Trc= 4, g-clear; Operand= ,
Trc= 5, g-ready, sig= 0 ; Operand= 0,
Trc= 6, g-ready, sig= 2 ; Operand= 0,
Trc= 7, g-ready, sig= 1 ; Operand= 0,
Trc= 8, g-begin, sig= 0 ; Operand= 0,
Trc= 9, g-begin, sig= 2 ; Operand= 0,
Trc= 10, g-begin, sig= 1 ; Operand= 0,
Trc= 11, g-begin, sig= 1107 ; Operand= 0,
Trc= 12, g-negated1, sig= 0 ; Operand= 0,
Trc= 13, g-begin, sig= 0 ; Operand= 0,
Trc= 14, g-calc, sig= 0 ; Operand= 0,
Trc= 15, g-begin, sig= 3 ; Operand= 0,

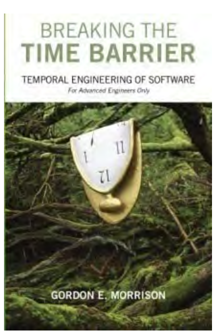
Trc= 16, g-ready, sig= 3 ; Operand= 0,
Trc= 17, g-ready, sig= 0 ; Operand= 0,
Trc= 18, g-negated1, sig= 2 ; Operand= 0,
Trc= 19, g-negated1, sig= 1 ; Operand= -0,
Trc= 20, g-negated1, sig= 100 ; Operand= -0,
Trc= 21, g-calc, sig= 100 ; Operand= -0,
Trc= 22, g-negated1, sig= 3 ; Operand= -0,
Trc= 23, g-final, sig= 0 ; Operand= -0,
- End of Analysis
Trc= 24, g-calc, sig= 0 ; Operand= -0,
Trc= 25, g-calc, sig= 3 ; Operand= -0,
Trc= 26, g-final, sig= 2 ; Operand= -0,
Trc= 27, g-final, sig= 1 ; Operand= -0,
- End of Analysis



A COSA State Diagram

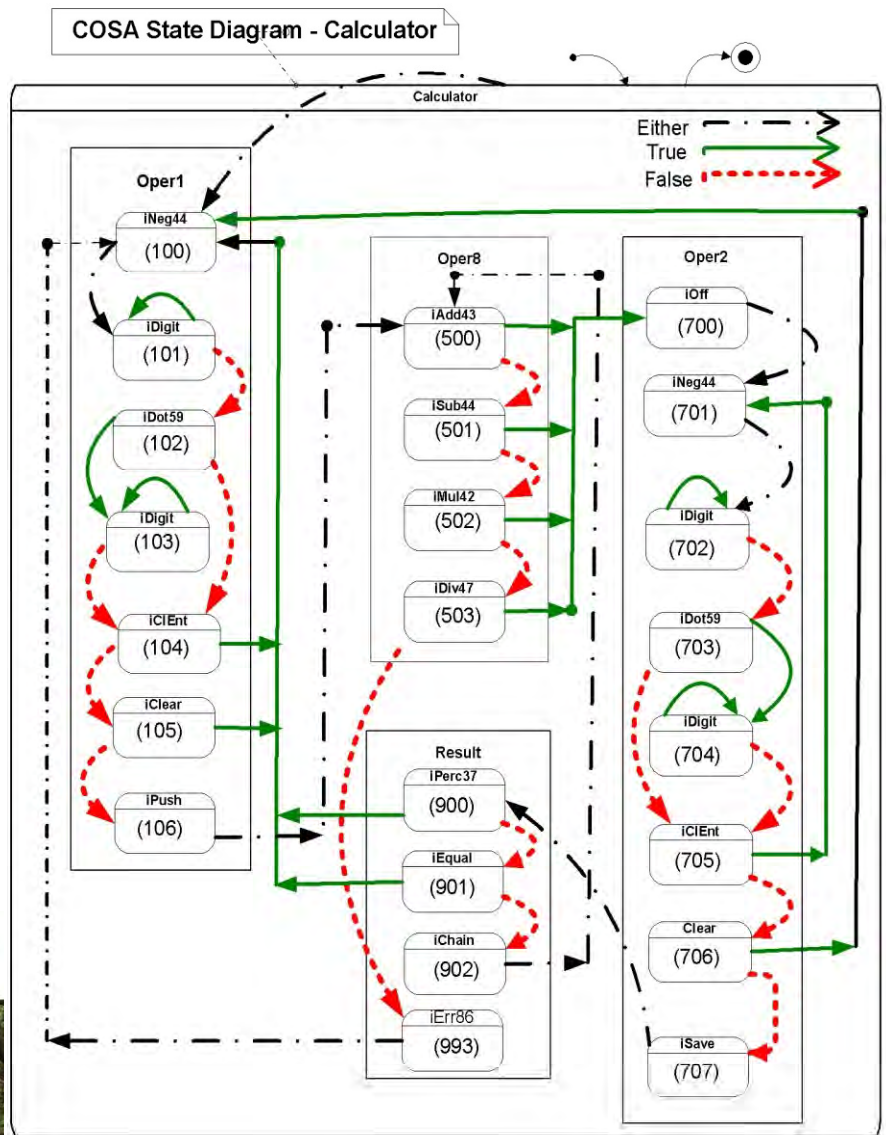
- Simple state view
- True Behavior
  - One green arrow
- False Behavior
  - One red arrow
- Temporal
  - Trace
  - Specification
    - Compliance

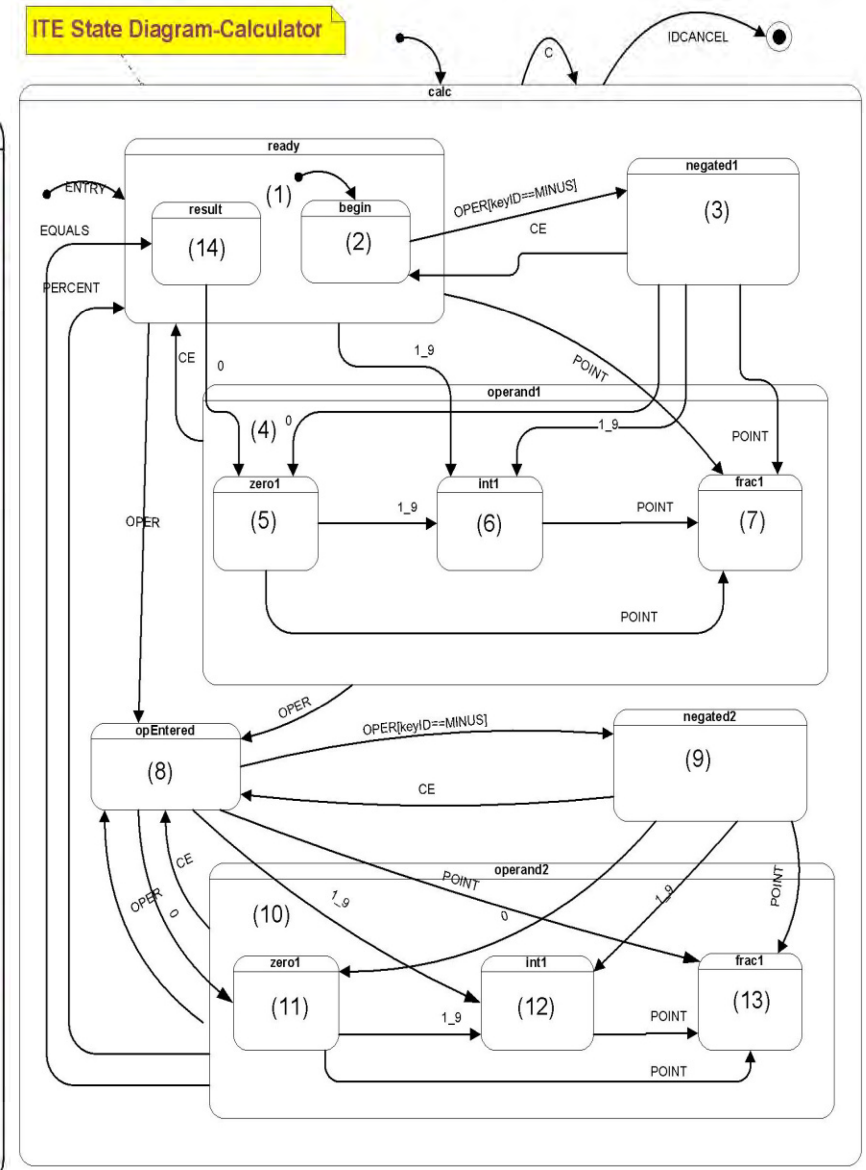


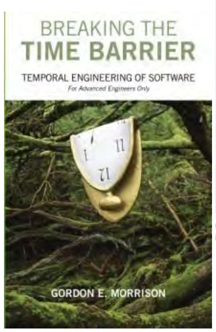


Statechart Comparison

COSA ITE

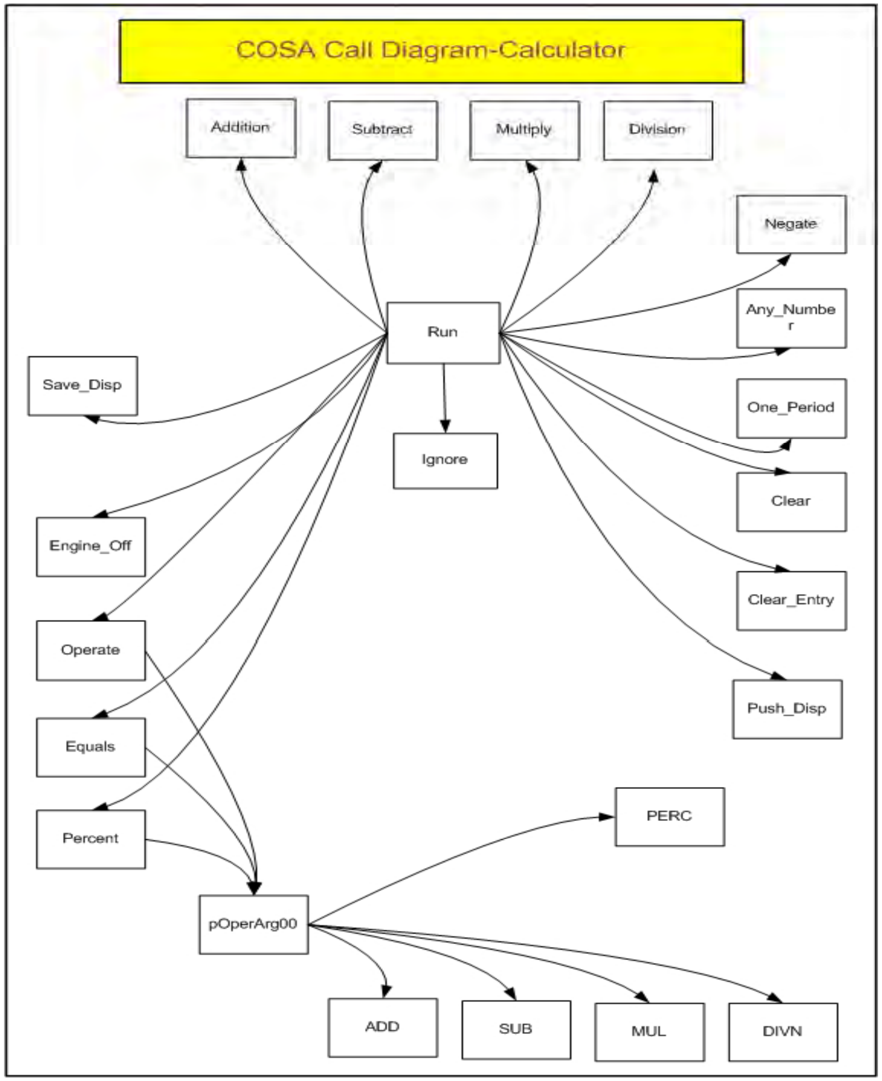


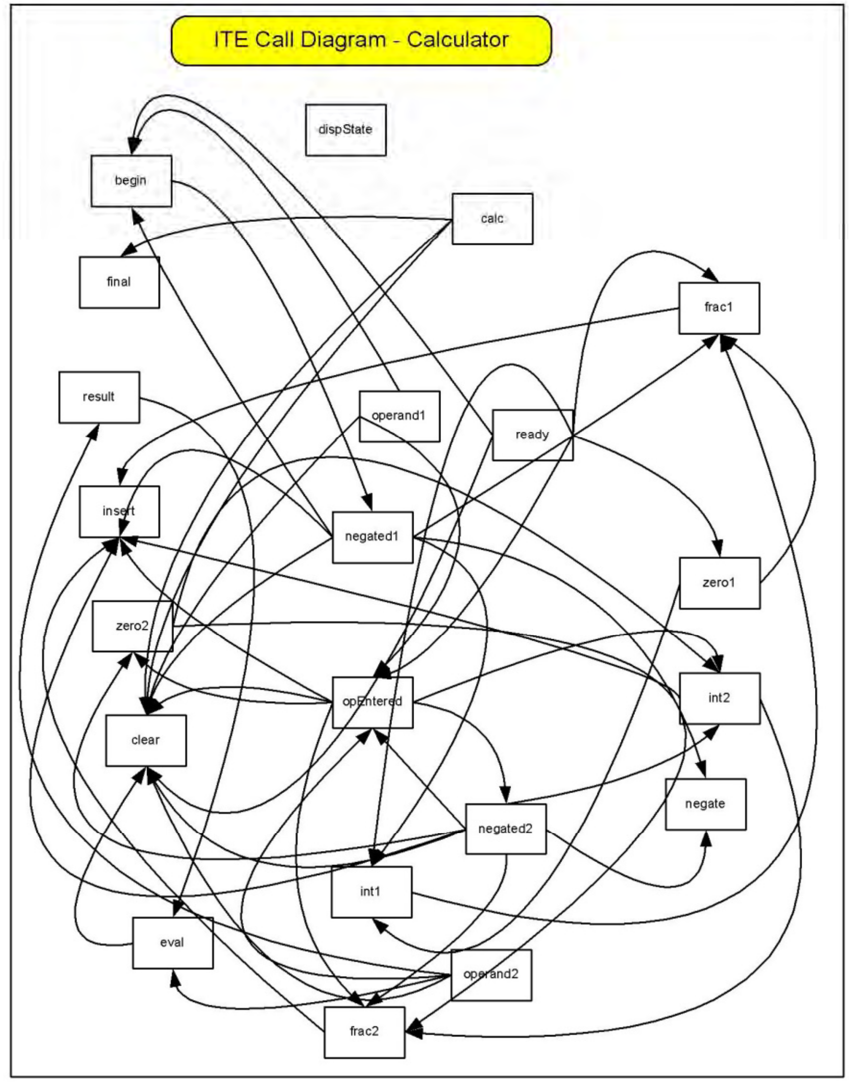


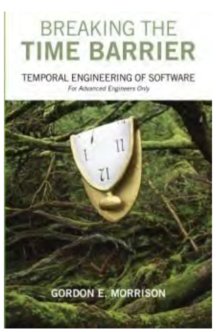


Call Diagram Complexity

COSA ITE

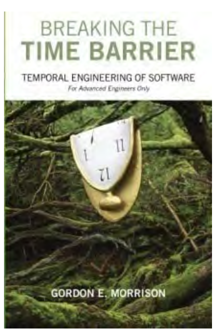






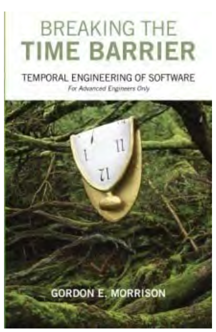
COSA and Time

- Understanding "Time" in software means not having to do an "if" to test **where** the program is executing and what has happened.
  - 23 Logic points in COSA calculator example
  - 112 IF statements in the ITE calculator example



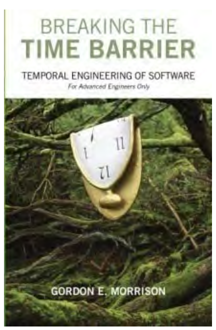
Spatial Software

- Must leave a trail of “breadcrumb” states
- Must track down where it was
  - This is pure overhead
- Difficult to maintain
- Difficult to modify



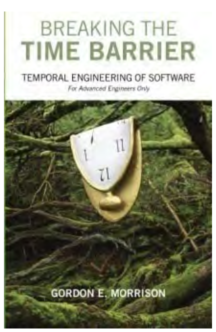
Temporal Software

- Keeps a temporal pointer
- Reduces complexity
- Eliminates much of the overhead
- Easier to maintain
- Easier to modify
  - Add new rule



Software Quality

- Testing doesn't improve quality
  - Testing fixes quality problems
  - Quality is still poor
- Temporal engineering
  - Improves quality
  - Reduces overhead logic



The End – Definitions

- COSA – Coherent Object System Architecture
  - U.S. Patent #6,345,387 abandoned by inventor
  - Available to the public in book: *Breaking the Time Barrier*
- BNF – Backus-Naur Format
  - Diagramming the logic of syntax
- ITE – If-Then-Else logic
  - commonly referred to as ‘spaghetti code’
- CMU – Carnegie Mellon University
- SEI – Software Engineering Institute at CMU
- CPU – Central Processing Unit
- UML – Unified Modeling Language